NUMERICAL RECIPES

Webnote No. 25, Rev. 1

Solvde Implementation

solvde.h

```
struct Solvde {
```
Driver routine for solution of two-point boundary value problems by relaxation.
```
    const Int itmax;
    const Doub conv;
    const Doub slowc;
    const VecDoub &scalv;
    const VecInt &indexv;
    const Int nb;
    MatDoub &y;
    Difeq &difeq;
    Int ne,m;
    VecInt kmax;
    VecDoub ermax;
    Mat3DDoub c;
    MatDoub s;
    Solvde(const Int itmaxx, const Doub convv, const Doub slowcc,
        VecDoub_I &scalvv, VecInt_I &indexvv, const Int nbb,
        MatDoub_IO &yy, Difeq &difeqq);
    void pinvs(const Int ie1, const Int ie2, const Int je1, const Int jsf,
        const Int jc1, const Int k);
    void bksub(const Int ne, const Int nb, const Int jf, const Int k1,
        const Int k2);
    void red(const Int iz1, const Int iz2, const Int jz1, const Int jz2,
        const Int jm1, const Int jm2, const Int jmf, const Int ic1,
        const Int jc1, const Int jcf, const Int kc);
};

Solvde::Solvde(const Int itmaxx, const Doub convv, const Doub slowcc,
    VecDoub_I &scalvv, VecInt_I &indexvv, const Int nbb, MatDoub_IO &yy,
    Difeq &difeqq) : itmax(itmaxx), conv(convv), slowc(slowcc),
    scalv(scalvv), indexv(indexvv), nb(nbb), y(yy), difeq(difeqq),
    ne(y.nrows()), m(y.ncols()), kmax(ne), ermax(ne), c(ne,ne-nb+1,m+1),
    s(ne,2*ne+1)
```
Constructor. `itmax` is the maximum number of iterations. `conv` is the convergence criterion (see text). `slowc` controls the fraction of corrections applied after each iteration. `scalv[0..ne-1]` contains typical sizes for each dependent variable, used to weight errors. `indexv[0..ne-1]` lists the column ordering of variables used to construct the matrix `s` of derivatives, defined internally. (The `nb` boundary conditions at the first mesh point must contain some dependence on the first `nb` variables listed in `indexv`.) The problem involves `ne` equations for `ne` adjustable dependent variables at each point. At the first mesh point there are `nb` boundary conditions. There are a total of `m` mesh points. `y[0..ne-1][0..m-1]` is the two-dimensional array that contains the initial guess for all the dependent variables at each mesh point. On each iteration, it is updated by the calculated correction.
```
{
    Int jv,k,nvars=ne*m;
    Int k1=0,k2=m;                              Set up row and column markers.
    Int j1=0,j2=nb,j3=nb,j4=ne,j5=j4+j1,j6=j4+j2,j7=j4+j3,j8=j4+j4,j9=j8+j1;
    Int ic1=0,ic2=ne-nb,ic3=ic2,ic4=ne,jc1=0,jcf=ic3;
    for (Int it=0;it<itmax;it++) {          Primary iteration loop.
        k=k1;                               Boundary conditions at first point.
        difeq.smatrix(k,k1,k2,j9,ic3,ic4,indexv,s,y);
        pinvs(ic3,ic4,j5,j9,jc1,k1);
        for (k=k1+1;k<k2;k++) {             Finite difference equations at all point pairs.
            Int kp=k;
            difeq.smatrix(k,k1,k2,j9,ic1,ic4,indexv,s,y);
            red(ic1,ic4,j1,j2,j3,j4,j9,ic3,jc1,jcf,kp);
            pinvs(ic1,ic4,j3,j9,jc1,k);
        }
        k=k2;                               Final boundary conditions.
        difeq.smatrix(k,k1,k2,j9,ic1,ic2,indexv,s,y);
        red(ic1,ic2,j5,j6,j7,j8,j9,ic3,jc1,jcf,k2);
        pinvs(ic1,ic2,j7,j9,jcf,k2);
        bksub(ne,nb,jcf,k1,k2);             Backsubstitution.
        Doub err=0.0;
        for (Int j=0;j<ne;j++) {            Convergence check, accumulate average er-
            jv=indexv[j];                       ror.
            Doub errj=0.0,vmax=0.0;
            Int km=0;
            for (k=k1;k<k2;k++) {           Find point with largest error, for each de-
                Doub vz=abs(c[jv][0][k]);       pendent variable.
                if (vz > vmax) {
                    vmax=vz;
                    km=k+1;
                }
                errj += vz;
            }
            err += errj/scalv[j];           Note weighting for each dependent variable.
            ermax[j]=c[jv][0][km-1]/scalv[j];
            kmax[j]=km;
        }
        err /= nvars;
        Doub fac=(err > slowc ? slowc/err : 1.0);
        Reduce correction applied when error is large.
        for (Int j=0;j<ne;j++) {            Apply corrections.
            jv=indexv[j];
            for (k=k1;k<k2;k++)
            y[j][k] -= fac*c[jv][0][k];
        }
        cout << setw(8) << "Iter.";         Summary of corrections for this step.
        cout << setw(10) << "Error" << setw(10) <<  "FAC" << endl;
        cout << setw(6) << it;
        cout << fixed << setprecision(6) << setw(13) << err;
        cout << setw(12) << fac << endl;    Point with largest error for each variable can
        if (err < conv) return;                 be monitored by writing out kmax and
    }                                           ermax arrays.
    throw("Too many iterations in solvde");           Convergence failed.
}

void Solvde::pinvs(const Int ie1, const Int ie2, const Int je1, const Int jsf,
    const Int jc1, const Int k)
```

Diagonalize the square subsection of the `s` matrix, and store the recursion coefficients in `c`; used internally by `Solvde`.

```
{
    Int jpiv,jp,je2,jcoff,j,irow,ipiv,id,icoff,i;
    Doub pivinv,piv,big;
    const Int iesize=ie2-ie1;
    VecInt indxr(iesize);
    VecDoub pscl(iesize);
    je2=je1+iesize;
    for (i=ie1;i<ie2;i++) {              Implicit pivoting, as in §2.1.
        big=0.0;
        for (j=je1;j<je2;j++)
```

```
            if (abs(s[i][j]) > big) big=abs(s[i][j]);
        if (big == 0.0)
            throw("Singular matrix - row all 0, in pinvs");
        pscl[i-ie1]=1.0/big;
        indxr[i-ie1]=0;
    }
    for (id=0;id<iesize;id++) {
        piv=0.0;
        for (i=ie1;i<ie2;i++) {         Find pivot element.
            if (indxr[i-ie1] == 0) {
                big=0.0;
                for (j=je1;j<je2;j++) {
                    if (abs(s[i][j]) > big) {
                        jp=j;
                        big=abs(s[i][j]);
                    }
                }
                if (big*pscl[i-ie1] > piv) {
                    ipiv=i;
                    jpiv=jp;
                    piv=big*pscl[i-ie1];
                }
            }
        }
        if (s[ipiv][jpiv] == 0.0)
            throw("Singular matrix in routine pinvs");
        indxr[ipiv-ie1]=jpiv+1;         In place reduction. Save column ordering.
        pivinv=1.0/s[ipiv][jpiv];
        for (j=je1;j<=jsf;j++) s[ipiv][j] *= pivinv;     Normalize pivot row.
        s[ipiv][jpiv]=1.0;
        for (i=ie1;i<ie2;i++) {         Reduce nonpivot elements in column.
            if (indxr[i-ie1] != jpiv+1) {
                if (s[i][jpiv] != 0.0) {
                    Doub dum=s[i][jpiv];
                    for (j=je1;j<=jsf;j++)
                        s[i][j] -= dum*s[ipiv][j];
                    s[i][jpiv]=0.0;
                }
            }
        }
    }
    jcoff=jc1-je2;                      Sort and store unreduced coefficients.
    icoff=ie1-je1;
    for (i=ie1;i<ie2;i++) {
        irow=indxr[i-ie1]+icoff;
        for (j=je2;j<=jsf;j++) c[irow-1][j+jcoff][k]=s[i][j];
    }
}

void Solvde::bksub(const Int ne, const Int nb, const Int jf, const Int k1,
    const Int k2)
```

Backsubstitution, used internally by `Solvde`.

```
{
    Int nbf=ne-nb,im=1;
    for (Int k=k2-1;k>=k1;k--) {        Use recurrence relations to eliminate remaining
        if (k == k1) im=nbf+1;          dependences.
        Int kp=k+1;
        for (Int j=0;j<nbf;j++) {
            Doub xx=c[j][jf][kp];
            for (Int i=im-1;i<ne;i++)
                c[i][jf][k] -= c[i][j][k]*xx;
        }
    }
    for (Int k=k1;k<k2;k++) {           Reorder corrections to be in column 1.
```

```
        Int kp=k+1;
        for (Int i=0;i<nb;i++) c[i][0][k]=c[i+nbf][jf][k];
        for (Int i=0;i<nbf;i++) c[i+nb][0][k]=c[i][jf][kp];
    }
}

void Solvde::red(const Int iz1, const Int iz2, const Int jz1, const Int jz2,
    const Int jm1, const Int jm2, const Int jmf, const Int ic1,
    const Int jc1, const Int jcf, const Int kc)
```

Reduce columns `jz1..jz2-1` of the `s` matrix, using previous results as stored in the `c` matrix. Only columns `jm1..jm2-1` and `jmf` are affected by the prior results. Used internally by `Solvde`.

```
{
    Int l,j,i;
    Doub vx;
    Int loff=jc1-jm1,ic=ic1;
    for (j=jz1;j<jz2;j++) {                        Loop over columns to be zeroed.
        for (l=jm1;l<jm2;l++) {                    Loop over columns altered.
            vx=c[ic][l+loff][kc-1];
            for (i=iz1;i<iz2;i++) s[i][l] -= s[i][j]*vx;     Loop over rows.
        }
        vx=c[ic][jcf][kc-1];
        for (i=iz1;i<iz2;i++) s[i][jmf] -= s[i][j]*vx;       Plus final element.
        ic += 1;
    }
}
```